COMP20003, Algorithms and Data Structures, Semester 2, 2024
Assignment 2 – Report

Kerui Huang – 1463475 - keruih@student.unimelb.edu.au

Executive summary

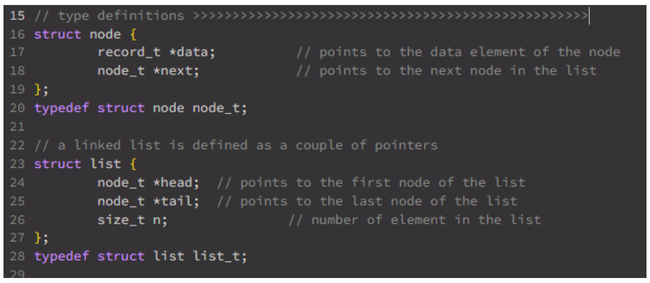
This paper aims to analyse the complexity of two data structures, linked lists and Patricia trees, used in a system for searching and storing records of suburbs provided by ABS data and edited by the COMP20003 team. The objective of this report is to evaluate and compare the efficiency of these structures when searching for keys of varying lengths across different data files through checking the number of key comparisons required by both data structures. How this performance aligns with theoretical expectations based on Big-O complexity is also focused on.

Introduction

A theoretical overview of the analysed data structures is as follows:

Linked List

A linked list structure is made up of a sequence of nodes, where each node contains the data (In this case, the suburb records) and a pointer to the next node in the list. In this implementation, the linked list itself will record the number of nodes, the head and tail (first and last node) of the list. Unlike arrays, where elements are stored contiguously, linked lists store elements in different memory locations, with each node pointing to the next. This allows for dynamic memory allocation and makes it easier to add or remove elements without requiring resizing or shifting elements.



```
// type definitions >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
struct node {
        record_t *data;         // points to the data element of the node
        node_t *next;           // points to the next node in the list
};
typedef struct node node_t;

// a linked list is defined as a couple of pointers
struct list {
        node_t *head;  // points to the first node of the list
        node_t *tail;  // points to the last node of the list
        size_t n;              // number of element in the list
};
typedef struct list list_t;
```

Figure 1: Linked list definitions in code

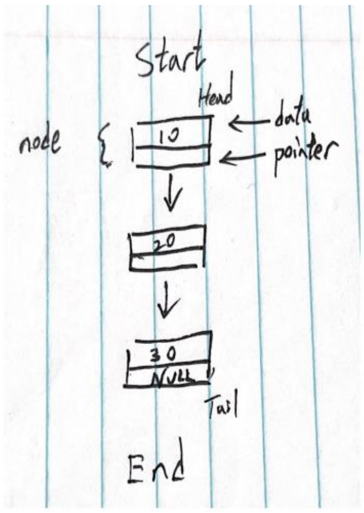


Figure 2: Illustration of a linked list structure

In a linked list, insertion is achieved through adjusting the pointers of existing nodes. If a new node is always inserted to the end of the list, then the pointer of the new node is added to the "next node" pointer of the last element in the list, after traversing through the entire linked list. Hence, the insertion complexity for a linked list would generally be O(n), where n is the number of elements. However, in this implementation, since the tail node is also recorded as shown in figure 1, the insert complexity would be O(1) in this case.

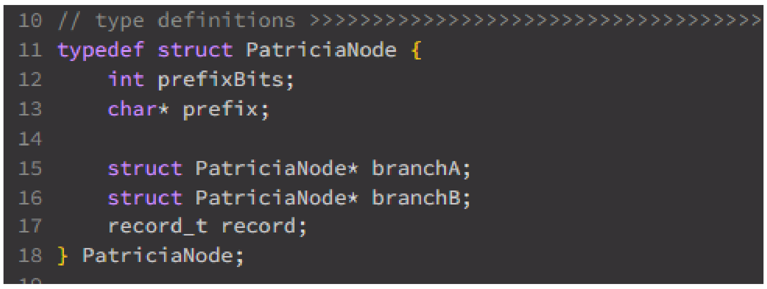
Similarly, the search operation for linked lists also requires traversing through the entire list. This means that the best-case complexity would be O(1), where the key is located at the start of the list, and the worst case complexity would be O(n) again, where the key is located at the end.

The space complexity of a linked list will generally be O(n), since it requires space for n nodes, where each node holds the key and pointer to the next node. In this case, since each node holds a struct for the suburb records, the space required will be proportional to this.

Patricia Trie (Tree)

A Patricia trie is a compressed version of a trie, and a type of Radix trie. Instead of storing every character in individual nodes, Patricia tries compress chains of single-child nodes, storing only common prefixes in bit level. Each internal node represents a decision based on the bits of the key, with two child branches for 0 and 1. In this implementation, the leaf nodes hold suburb records, and the traversal is based on bit comparisons with node prefixes. Compared to a standard trie, the patricia tree is compact and has fewer nodes, making it more space-efficient.

In this project, it is specified to add a "spell checker" for the search function of a patricia tree, which uses the edit distance of strings to find the closest match. This will not be taken into account for the analysis of complexities since it introduces more comparisons and accesses external to the patricia tree structure, which doesn't exist for the linked list structure.



```c
// type definitions >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
typedef struct PatriciaNode {
    int prefixBits;
    char* prefix;

    struct PatriciaNode* branchA;
    struct PatriciaNode* branchB;
    record_t record;
} PatriciaNode;
```

Figure 3: Patricia trie definitions in code

In a Patricia tree, insertion is done through following the key's bits until a mismatch is found, then adding it as a new node (unless it is a duplicate, in which case it gets stored as a duplicate with the same leaf node). Therefore in this case, the worst case insertion complexity is O(k) and the best case is O(l), where k equals the entire length of the inserted key and l equals the length of the longest common prefix.

Similarly, the search operation is achieved through matching the key's bits with the prefix's bits of the existing nodes, and it involves bitwise traversal along the tree. Hence, the search complexity of a Patricia tree is also theoretically O(k), with k being the length of the searched key.

The space complexity of a Patricia tree will depend on the similarities of prefix bits of the nodes, ranging from O(n) to O(k*n) non inclusive, where n is the number of nodes and k is the length of each prefix (best case is when all keys share long common prefixes and minimal branching occurs, worst case is the opposite). Generally, the complexity is O(b), where b represents the total number of bits required to store all the keys. Again, since each leaf node holds a struct for the suburb records, the space required for the leaf nodes will be proportional to this.

Space-time tradeoff

Since the space complexities of a Patricia tree will be generally larger than linked lists, a space-time trade off exists for using this structure. The Patricia tree provides significantly better search times than linked lists but use more memory per node. On the other hand, linked lists use minimal space (each node containing data and a pointer), but is inefficient in searching, having to use linear search for the key. Hence it is understood that their space efficiency generally suits small datasets but not large ones where search time becomes prohibitive, in which case Patricia trees can be used.

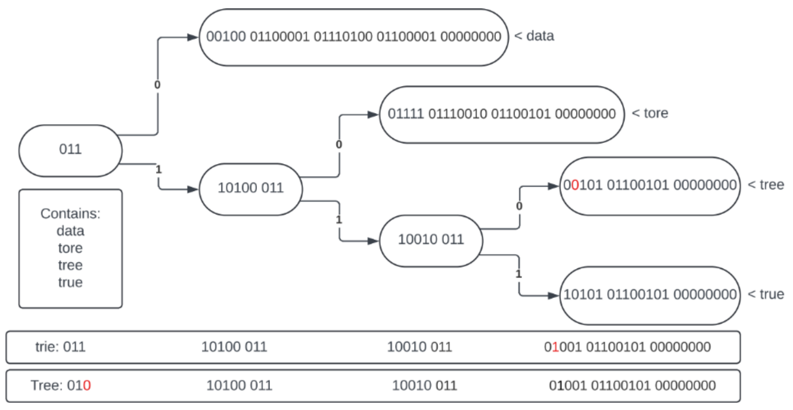


Figure 4: The search process in an example Patricia trie

Hypothesis

Given the theory, it is predicted that the Patricia Trie data structure will be overall more efficient than linked lists and require less operations and times for inserting and searching. In certain situations where the key lengths are very long and share few common prefixes, linked lists could be more efficient. However, generally, linked lists are expected to be less efficient as the dataset size grows, as its complexity relies on the number of nodes.

Methodology

Metrics Measured

To measure and compare the efficiency of the data structures, the following metrics are used as instructed by the Project Implementation details. Using the same metrics ensures consistency when evaluating the performance of both structures.

- Bit Comparisons:
  (Each character compared adds exactly 8 bits to the bit comparison count for linked lists)

- Node Accesses
  (The node access is incremented once per accessed linked list / trie node)

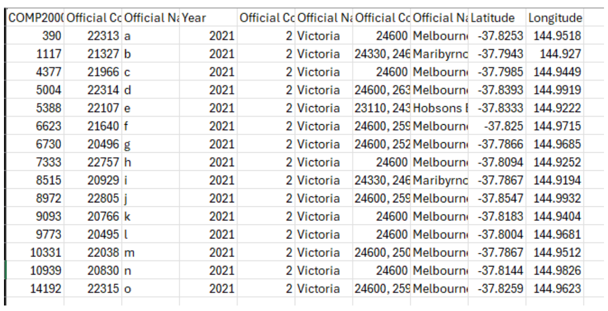
- String Comparisons
  (Each string comparison, even if a mismatch occurs on the first character, is 1 string comparison)

Test Data

3 sets of varying testing data are prepared for testing the data structures on their efficiency/complexity, with each having a particular focus. Additionally, a control data is also prepared and intended to be used for effective comparisons. The datasets are outlined below:

Control

To create a dataset with little bias, which can be used for comparisons against other test cases, the provided test_15.csv file is edited with the key value column "OfficialNameSuburb" replaced with 15 alphabet letters. Since these values don't have common prefixes (in characters), it ensures that Patricia trees won't benefit from extra shared bits, allowing for a more balanced comparison with linked lists in terms of search and insertion efficiency.

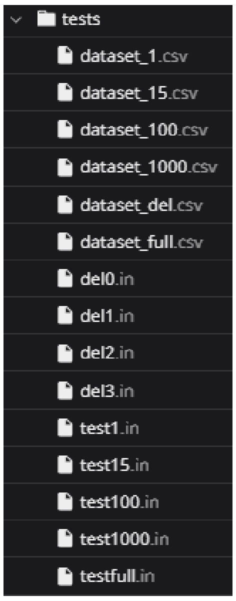


COMP200	Official Cc	Official Na	Year	Official Cc	Official Na	Official Cc	Official Na	Latitude	Longitude
390	22313	a	2021	2	Victoria	24600	Melbourn	-37.8253	144.9518
1117	21327	b	2021	2	Victoria	24330, 246	Maribyrnc	-37.7943	144.927
4377	21966	c	2021	2	Victoria	24600	Melbourn	-37.7985	144.9449
5004	22314	d	2021	2	Victoria	24600, 263	Melbourn	-37.8393	144.9919
5388	22107	e	2021	2	Victoria	23110, 243	Hobsons I	-37.8333	144.9222
6623	21640	f	2021	2	Victoria	24600, 259	Melbourn	-37.825	144.9715
6730	20496	g	2021	2	Victoria	24600, 252	Melbourn	-37.7866	144.9685
7333	22757	h	2021	2	Victoria	24600	Melbourn	-37.8094	144.9252
8515	20929	i	2021	2	Victoria	24330, 246	Maribyrnc	-37.7867	144.9194
8972	22805	j	2021	2	Victoria	24600, 259	Melbourn	-37.8547	144.9932
9093	20766	k	2021	2	Victoria	24600	Melbourn	-37.8183	144.9404
9773	20495	l	2021	2	Victoria	24600	Melbourn	-37.8004	144.9681
10331	22038	m	2021	2	Victoria	24600, 250	Melbourn	-37.7867	144.9512
10939	20830	n	2021	2	Victoria	24600	Melbourn	-37.8144	144.9826
14192	22315	o	2021	2	Victoria	24600, 259	Melbourn	-37.8259	144.9623

Figure 5: Control data

Dataset Size

One of the most important components to this investigation is the input dataset size, since it directly influences the number of nodes, and potentially the complexity of the traversal of the structures. To test the data structures on a variety of Data sizes, the provided dataset_1.csv to dataset_full.csv will be utilised and inputted to both structures. These datasets range from 1 to 15334 records. With each dataset, the suburb "Carlton" will be inputted to be searched. It is noted that using Carlton as the search key can have preliminary biases (For instance, if the dataset tends to have more Suburbs with names beginning with Carlton or if the record for Carlton is inserted earlier in the linked list). For the sake of generalisation, this will be neglected.

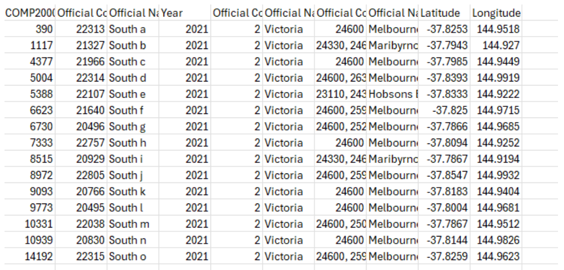


```
tests
    dataset_1.csv
    dataset_15.csv
    dataset_100.csv
    dataset_1000.csv
    dataset_del.csv
    dataset_full.csv
    del0.in
    del1.in
    del2.in
    del3.in
    test1.in
    test15.in
    test100.in
    test1000.in
    testfull.in
```

Figure 6: Data size testing

Prefix Similarity

To test if keys having common prefixes affect the efficiency of the two data structures, a custom dataset edited using the control is prepared. In this, the word "South" is added to the start of each key in the "officialNameSuburb" column. This is an attempt to ensure that there is at least one long common prefix in the Patricia Trie, in order to test its performance and compare it against the Linked List.

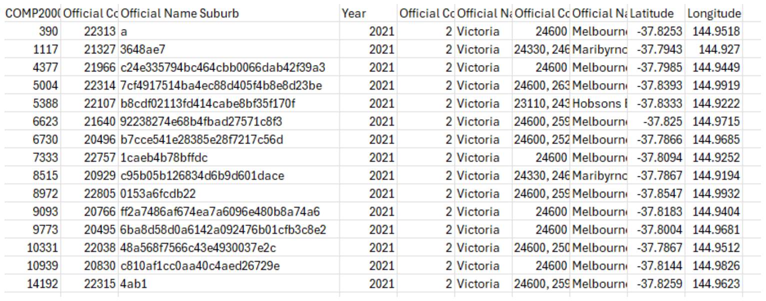


COMP200	Official Cc	Official Na	Year	Official Cc	Official Na	Official Cc	Official Na	Latitude	Longitude
390	22313	South a	2021	2	Victoria	24600	Melbourn	-37.8253	144.9518
1117	21327	South b	2021	2	Victoria	24330, 246	Maribyrnc	-37.7943	144.927
4377	21966	South c	2021	2	Victoria	24600	Melbourn	-37.7985	144.9449
5004	22314	South d	2021	2	Victoria	24600, 263	Melbourn	-37.8393	144.9919
5388	22107	South e	2021	2	Victoria	23110, 243	Hobsons I	-37.8333	144.9222
6623	21640	South f	2021	2	Victoria	24600, 259	Melbourn	-37.825	144.9715
6730	20496	South g	2021	2	Victoria	24600, 252	Melbourn	-37.7866	144.9685
7333	22757	South h	2021	2	Victoria	24600	Melbourn	-37.8094	144.9252
8515	20929	South i	2021	2	Victoria	24330, 246	Maribyrnc	-37.7867	144.9194
8972	22805	South j	2021	2	Victoria	24600, 259	Melbourn	-37.8547	144.9932
9093	20766	South k	2021	2	Victoria	24600	Melbourn	-37.8183	144.9404
9773	20495	South l	2021	2	Victoria	24600	Melbourn	-37.8004	144.9681
10331	22038	South m	2021	2	Victoria	24600, 250	Melbourn	-37.7867	144.9512
10939	20830	South n	2021	2	Victoria	24600	Melbourn	-37.8144	144.9826
14192	22315	South o	2021	2	Victoria	24600, 259	Melbourn	-37.8259	144.9623

Figure 7: Similar prefix testing data

Prefix Lengths (and variety)

In contrast, to test if keys having long, varied prefixes can affect the efficiency of the two data structures, another custom dataset edited from the control is prepared. In this, each key in the "officialNameSuburb" column is replaced with a UUID (Universally Unique Identifier) sequence of random lengths. This is an attempt to ensure that the Patricia Trie will have long node prefixes, given each key's uniqueness, in order to further test and compare the performance of the structures.



COMP200(	Official Cc	Official Name Suburb	Year	Official Cc	Official Na	Official Cc	Official Na	Latitude	Longitude
390	22313	a	2021	2	Victoria	24600	Melbourn	-37.8253	144.9518
1117	21327	3648ae7	2021	2	Victoria	24330, 246	Maribyrnc	-37.7943	144.927
4377	21966	c24e335794bc464cbb0066dab42f39a3	2021	2	Victoria	24600	Melbourn	-37.7985	144.9449
5004	22314	7cf4917514ba4ec88d405f4b8e8d23be	2021	2	Victoria	24600, 263	Melbourn	-37.8393	144.9919
5388	22107	b8cdf02113fd414cabe8bf35f170f	2021	2	Victoria	23110, 243	Hobsons I	-37.8333	144.9222
6623	21640	92238274e68b4fbad27571c8f3	2021	2	Victoria	24600, 259	Melbourn	-37.825	144.9715
6730	20496	b7cce541e28385e28f7217c56d	2021	2	Victoria	24600, 252	Melbourn	-37.7866	144.9685
7333	22757	1caeb4b78bffdc	2021	2	Victoria	24600	Melbourn	-37.8094	144.9252
8515	20929	c95b05b126834d6b9d601dace	2021	2	Victoria	24330, 246	Maribyrnc	-37.7867	144.9194
8972	22805	0153a6fcdb22	2021	2	Victoria	24600, 259	Melbourn	-37.8547	144.9932
9093	20766	ff2a7486af674ea7a6096e480b8a74a6	2021	2	Victoria	24600	Melbourn	-37.8183	144.9404
9773	20495	6ba8d58d0a6142a092476b01cfb3c8e2	2021	2	Victoria	24600	Melbourn	-37.8004	144.9681
10331	22038	48a568f7566c43e4930037e2c	2021	2	Victoria	24600, 250	Melbourn	-37.7867	144.9512
10939	20830	c810af1cc0aa40c4aed26729e	2021	2	Victoria	24600	Melbourn	-37.8144	144.9826
14192	22315	4ab1	2021	2	Victoria	24600, 259	Melbourn	-37.8259	144.9623

Figure 8: Prefix variance testing data

Generated using https://www.uuidgenerator.net/

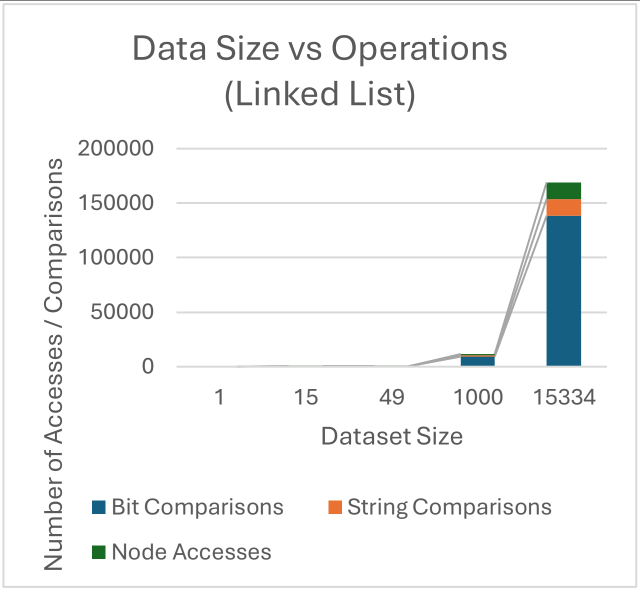
Results and Discussion

Data set size

The efficiency of both data structures, the linked list and the Patricia tree, was tested across various data sizes using the test data. By systematically increasing the number of records, the performance of how each structure scales is observed. Key metrics were tracked to evaluate the performance under different loads. The results are as follows: (*Note: the recorded metrics are solely from own implementation of the data structures).

Linked List Carlton			
Data Size	Bit Comparisons	String Comparisons	Node Accesses
1	64	1	1
15	232	15	15
49	568	49	49
1000	9296	1000	1000
15334	138512	15334	15334

Patricia Trie Carlton			
Data Size	Bit Comparisons	String Comparisons	Node Accesses
1	64	1	1
15	57	4	1
49	57	7	1
1000	57	15	1
15334	57	21	1



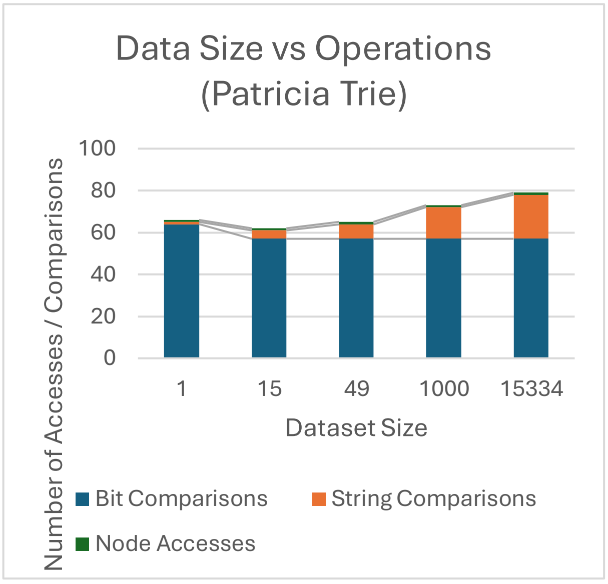


Figure 9: Data size results

As shown in figure 9 , the linked list implementation grew in the number of total operations as the dataset grew in size. The node accesses and string comparisons all equal the number of elements in the dataset, while the number of bit comparisons appears to scale with the data size, increase at a similar rate to the data size. This is within expectation, as this implementation of the linked list traverses and searches through every node of the list, to catch the matches including duplicates.

In contrast, the data size appears to have little impact on the number of total operations, despite its dramatic increase. In the recorded results, the Bit comparisons are shown to maintain a value of around 57, while the node accesses equals 1 consistently. Only the String comparisons appear to increase, which is explained by the "spell checker" component of this implementation. Otherwise, the results are also within expectations, as the search function only needs to match the prefixes of the trie and the key to find the matching node, which wouldn't scale with the data set size.

The relationship between the operation counts and the data structures are further displayed below. Since the linked list operations scales with the data size, and the Patricia trie doesn't, the results here are expected.

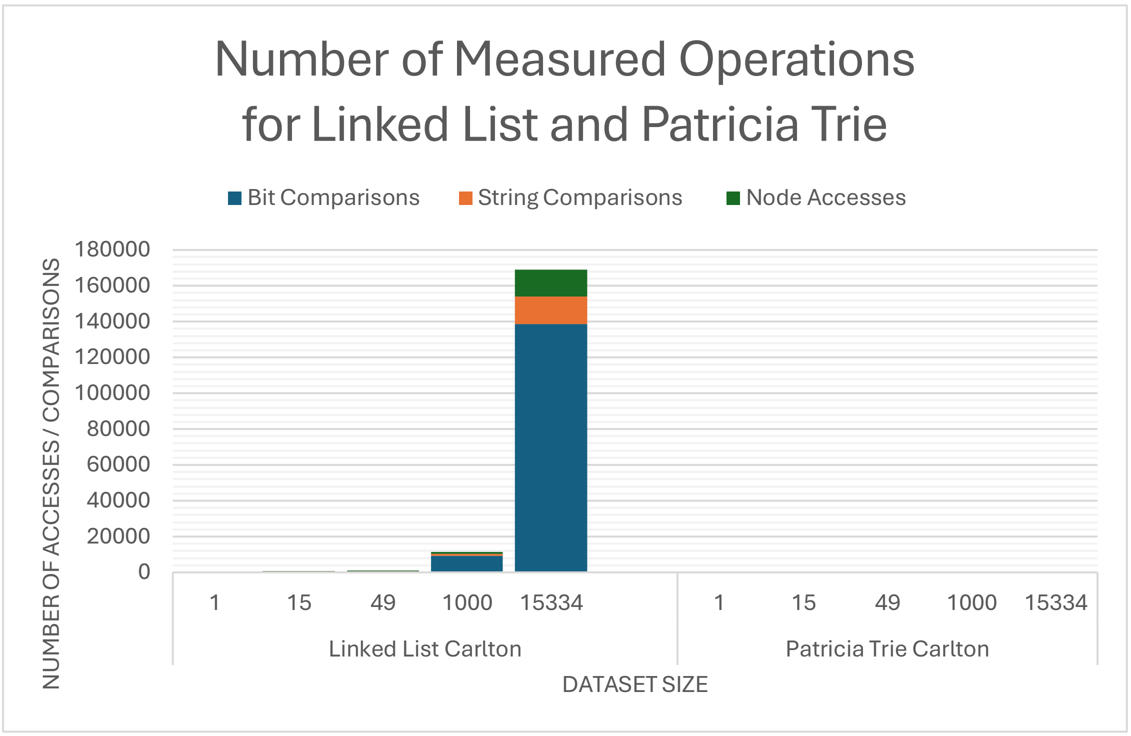
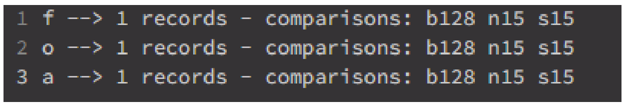
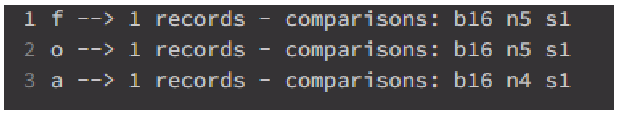


Figure 10: Data size results comparison

For the next part of the testings, the control results are displayed here for reference:



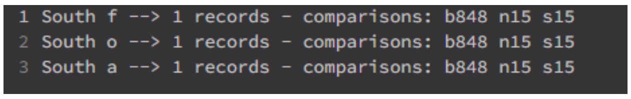
```
f --> 1 records - comparisons: b128 n15 s15
o --> 1 records - comparisons: b128 n15 s15
a --> 1 records - comparisons: b128 n15 s15
```



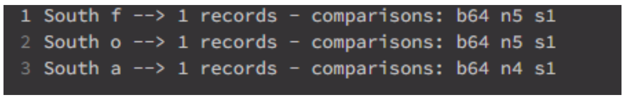
```
f --> 1 records - comparisons: b16 n5 s1
o --> 1 records - comparisons: b16 n5 s1
a --> 1 records - comparisons: b16 n4 s1
```

Figure 11: Control data results

Prefix Similarity



```
South f --> 1 records - comparisons: b848 n15 s15
South o --> 1 records - comparisons: b848 n15 s15
South a --> 1 records - comparisons: b848 n15 s15
```



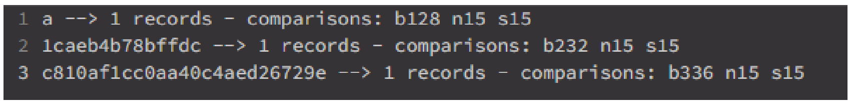
```
South f --> 1 records - comparisons: b64 n5 s1
South o --> 1 records - comparisons: b64 n5 s1
South a --> 1 records - comparisons: b64 n4 s1
```

Figure 12: Similar prefix data results

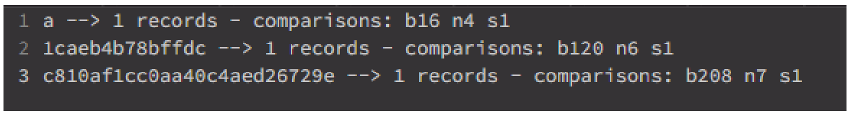
With a common prefix shared by all nodes, the bit comparisons increased for both data structures, which makes logical sense as the actual length of the keys also increased. The linked list bit comparisons all increased by 720, which matches the theoretical calculations. 6 characters were added for every node (15 nodes in total) Since every node is checked for this implementation of linked list, 6*15 bytes = 6*8*15 bits = 720 additional bits are required for searching.

The bit comparisons for the search in the trie on the other hand unanimously increased by 48, which also matches theoretical calculations. With 6 common characters added, the first prefix of the trie will therefore be elongated, including 6 additional bytes, or 6*8 = 48 bits. Since the search function only needs to traverse the prefixes once in the Patricia tree, the results are within calculation.

Prefix lengths and variety



```
a --> 1 records - comparisons: b128 n15 s15
1caeb4b78bffdc --> 1 records - comparisons: b232 n15 s15
c810af1cc0aa40c4aed26729e --> 1 records - comparisons: b336 n15 s15
```



```
a --> 1 records - comparisons: b16 n4 s1
1caeb4b78bffdc --> 1 records - comparisons: b120 n6 s1
c810af1cc0aa40c4aed26729e --> 1 records - comparisons: b208 n7 s1
```

Figure 13: Prefix length and variety data results

On the other hand, with data using mostly random and unsimilar key values, it is expected that the difference in bit operations between Patricia trees and Linked lists will be less than usual, since the trie will need to store more of the key in the prefix of the leaf node (rather than the internal node).

The resulting metric operations count for linked lists is shown above. The linked lists again show 15 node accesses and string comparisons, matching the node count of the list. The bit comparison count here theoretically depends on the position of the key in the list and the length of the keys. The bit comparisons of the Patricia trie is indeed larger than usual, relative to the linked list. However, the trie still has less operations overall.

Conclusion

This report demonstrates that the Patricia Trie generally consistently outperforms the linked list in terms of efficiency, particularly as the dataset size increases. Theoretically, the linked list's complexity grows linearly with the number of elements, as each search requires traversing through the entire list. This results in a significant increase in node accesses and bit comparisons as the dataset grows, making linked lists less efficient for large datasets. In contrast, the Patricia Trie's bit comparisons and node accesses remain relatively constant regardless of dataset size due to its ability to skip over common prefixes.

The hypothesis was largely supported: Patricia Tries, in general, proved more efficient for searching tasks, especially with larger datasets. However, the specific scenario of long, unique keys with few common prefixes showed that linked lists could reduce the Patricia Trie's typical performance advantages.

Thus, also considering the space-time trade off, Patricia Tries are more scalable and efficient structures for datasets with shared prefixes or large data sizes, while linked lists can still be suitable for small datasets or highly unique key sets.

References

GeeksforGeeks. (2021). *Implementing Patricia Trie in Java*. [online] Available at:https://www.geeksforgeeks.org/implementing-patricia-trie-in-java/.

GeeksforGeeks. (2024). *Time and Space Complexity of Linked List*. [online] Available at: https://www.geeksforgeeks.org/time-and-space-complexity-of-linked-list/.

guidgenerator.com. (n.d.). *Free Online GUID Generator*. [online] Available at: https://guidgenerator.com/.

Hatim (2023). [online] Tutorchase.com. Available at: https://www.tutorchase.com/answers/a-level/computer-science/what-is-a-patricia-tree--and-how-does-it-work.

Osiek, B. (2020). *PATRICIA Trie's Nuts and Bolts*. [online] Medium. Available at: https://medium.com/@brunoosiek/patricia-tries-nuts-and-bolts-170a317b3ab6.

Patricia (2016). *What is the difference between radix trees and Patricia tries?* [online] Computer Science Stack Exchange. Available at: https://cs.stackexchange.com/questions/63048/what-is-the-difference-between-radix-trees-and-patricia-tries [Accessed 6 Sep. 2024].

Ravikiran A S (n.d.). *Linked List in a Data Structure: All You Need to Know*. [online] Simplilearn.com. Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/linked-list-in-data-structure#:~:text=A%20linked%20list%20is%20the.